Chalmers University of Technology and Gothenburg University

Operating Systems
EDA093, DIT 401
Exam 2019-01-07

Date, Time, Place: Monday 2019/01/07, 08.30-12.30, Samhållsbyggnad

Course Responsible:
Vincenzo Gulisano (031 772 61 47),
Marina Papatriantafilou (031 772 54 13)

Auxiliary material: You may have with you
- An English-Swedish, Swedish-English dictionary.
- No other books, notes, calculators, etc.

Grade-scale ("Betygsgränser"):
CTH: 3:a 30-39 p, 4:a 40-49 p, 5:a 50-60 p
GU: Godkänd 30-49p, Väl godkänd 50-60 p

Exam review ("Granskningstid"):
Will be announced after the exam.

Instructions

- Do not forget to write your personal number, if you are a GU or CTH student and at which program ("linje").
- Start answering each assignment on a new page; number the pages and use only one side of each sheet of paper.
- Write in a **clear manner** and **motivate** (explain, justify) your answers. If it is not clear what is written, your answer will be considered wrong. If it is not explained/justified, even a correct answer will get **significantly** lower (possibly zero) marking.
- If you make **any assumptions** in answering any item, do not forget to clearly state what you assume.
- The exam is organized in groups of questions. The credit for each group of questions is mentioned in the beginning of the respective group. Unless otherwise stated, all questions in a group have equal weight.
- Answer questions in English, if possible. If you have large difficulty with that and you think that your grade can be affected, feel free to write in Swedish.

Good luck !!!!

1. (12 p)

   (a) (4 p) If the serial portion of a program is 0.25, find the number of cores needed to run such program in parallel and achieve a speed-up of 25.
   [**HINT:** cannot go more than or equal to 4 times faster.]

   (b) (4 p) Motivate why a certain application would leverage both multiple processes as well as multiple threads to run in parallel and concurrently in a multi-core architecture.
   [**HINT:** e.g. for modularity (processes) as well as shared state (threads) for some functionality.]

   (c) (4 p) What happens to the threads of a process when *exec* is called? Why?
   [**HINT:** Threads are destroyed.]

2. (12 p)

   (a) (4 p) Provide examples of how virtual memory with paging decouples the physical view of memory from the logical one.
   [**HINT:** e.g. addresses starting at 0. Solving external fragmentation by means of pages. Loading only parts of a program. Running a program even if such program cannot fit entirely in memory...]

   (b) (4 p) Discuss a scenario in which a certain process can experience page faults because of other processes.
   [**HINT:** When pages of such process are replaced to load other processes' pages (global replacement).]

   (c) (4 p) How does copy-on-write speed up the execution of a *fork* call?
   [**HINT:** By skipping the copy of pages that can be later read by the new process (or replaced by exec later on).]

3. (12 p)

   (a) (4 p) Explain what Direct Memory Access is and why it can improve CPU utilization.
   [**HINT:** Please refer to the slides of the corresponding lecture (I/O System)]

   (b) (4 p) Discuss why the timer interrupt is a key component of a modern OS.
   [**HINT:** e.g., marking the end of each quantum to support the scheduler.]

   (c) (4 p) Discuss how the information is stored on disk and accessed by the hardware of a computer during the booting process when multiple OSes are installed.
   [**HINT:** Please refer to the slides of the corresponding lecture (File Systems). In this case a boot loader is needed.]

4. (12 p)

   (a) (4p) Explain the role of (i) synchronization and (ii) memory organization, in making scheduling in multiprocessor/multicore systems a more complex problem compared to single-processor scheduling.
   [**HINT:** Both factors introduce issues making CPU utilization less inline with goals for progress of threads and of the system in general. Threads that synchronize frequently can benefit from being dispatched simultaneously, making it sometimes less important to interleave threads

of different processes in the same CPU. Migration of state in NUMA systems can introduce more overhead than savings when trying to share load (i.e. processor affinity can be beneficial sometimes, even if there exist underutilized CPUs)]

(b) (4 p) Consider three periodic tasks, T1, T2 and T3, with periods and processing requirements of p1 = 60, t1 = 20, p2 = 30, t2 = 5, p3 = 120, and t3 = 40, respectively. (i) Can they be scheduled using rate-monotonic scheduling? (ii) Can they be scheduled using the earliest-deadline-first (EDF) scheduling? Explain carefully why or why not in both questions.

[**HINT:** RM cannot do it; apply it and show with Gantt chart that some task misses its deadline. EDF can do it; we know EDF can schedule all task sets where the sum of ti/pi does not exceed 1 and this is the case here.]

(c) (4 p) The *regressive round-robin scheduler* assigns each schedulable unit (i.e. each process or thread; for simplicity we only refer to threads in the rest of this question) a time quantum and a priority. The initial value of a time quantum and priority are fixed, e.g. 50 msec and level 0. Threads are dispatched in a round robin fashion, starting from the highest priorities and moving to the lower ones, with each thread being allocated the CPU for a time interval of length at most its time quantum. Every time a dispatched thread uses its entire time quantum (i.e. does not block for I/O), its time quantum is increased (by e.g. 10 msec) and its priority level is boosted, up to a maximum of e.g. 100 msec and level 5. When a process blocks before using its entire time quantum, the latter is reduced (by e.g. 5 msec), but its priority remains the same. What data structure can be used for the ready queue? What type of threads (CPU-bound or I/O-bound) does this scheduler favor? Explain both parts of your answer.

[**HINT:** CPU bound threads get higher priority and higher quantum, hence they are favoured. Multilevel queue or sorted linked list can be used, sorting threads after their priority.]

5. (12 p)

(a) (4 p) Servers can be designed to limit the number of open connections. For example, a server may wish to have only N active socket connections at any point in time. As soon as N connections are made, the server will not accept a new incoming connection until an existing one is released. Describe how semaphores can be used to limit the number of concurrent connections; use pseudocode and also explain your answer.

[**HINT:** General semaphore S, initialized to N; execute wait(S) and signal(s) before and after the connection between each client (socket) and server.]

(b) (4 p) Show a method that solves the critical section problem for arbitrary number of threads using the atomic TestAndSet instruction that is available in several processor architectures. Use pseudocode in the description and argue about the properties of the solution, with respect to mutual exclusion, progress and fairness. (It is not necessary to describe a solution that guarantees fairness in this question, but if you can, of course it is ok).

[**HINT:** Pseudocode slides 17 (or 46, with fairness) in synchronization lecture. Argumentation as for the mutual exclusion algorithm by Peterson.]

(c) (4 p)
(a) Name and describe the four necessary conditions for a deadlock to occur among threads/processes, in the context of allocation of reusable resources.
(b) How does the knowledge of these conditions help in preventing deadlocks? Explain carefully your answer.
[**HINT:** Mutual exclusion, circular wait, no preemption, hold and wait. If one of them is not possible in a RA method, then deadlock is prevented.]